Optimization Algorithm Inspired by Social Insect Behaviour in Comparison with Hill Climbing

Daniel Soós*
Institute of Applied Informatics
Faculty of Informatics and Information Technologies
Slovak University of Technology in Bratislava
Ilkovičova 3, 842 16 Bratislava, Slovakia
soos.dano@gmail.com

Abstract

This paper experiments with an algorithm inspired by the social insect behaviour. In this method, each member of the population represents a solution to the optimization problem and the algorithm can be described as a multiagent method. Lifespans of these agents are based on the quality of the solution. One agent in every generation moves out of the nest as it seeks for food in the artificial world. Afterwards, if the case is that it found food, other agents staying in the nest will know about the food source. New solutions are generated each generation thanks to mutation. We test this algorithm in comparison with a stochastic parallel hill climbing algorithm on a typical non-convex function, the Rastrigin function and other well-known mathematic functions which we want to minimize. Our results show that the newly proposed algorithm outperforms the parallel hill climbing method on harder tasks, when we tested both algorithms on more than one-dimensional functions. There is also room for improvement in the presented algorithm and we suggest a new technique built into it that may work properly.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Heuristic methods*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*

*Bachelor degree study programme in field Software Engineering. Supervisor: David Chalupa, Institute of Informatics and Software Engineering, Faculty of Informatics and Information Technologies, STU in Bratislava.
Work described in this paper was presented at the 9th Student Research Conference in Informatics and Information Technologies IIT.SRC 2013 and awarded by Dean's Award.

Soós, D. Optimization Algorithm Inspired by Social Insect Behaviour in Comparison with Hill Climbing. Information Sciences and Technologies Bulletin of the ACM Slovakia, Special Section on Student Research in Informatics and Information Technologies, Vol. 5, No. 2 (2013) 48-52

Keywords

Bumblebees Algorithm, Optimization, Social Insect Behaviour, Parallel Hill Climbing

1. Introduction

Finding global minimas for non-convex functions with several local minimas is a computationally hard task if we decide not to use stochastic methods to find the global solution. A few algorithms and heuristics were designed for such problems. For example, the approach of artificial intelligence based on the social insect behaviour produced very efficient results in the last few years in some types of optimization problems [9]. In this work, we test an algorithm which is a metaphore to the food acquisition of a bumblebee community. Our evaluation of this algorithm is reached by a comparison with a stochastic parallel hill climbing algorithm. The original aim of this paper was to contribute on some improvement on two selected non-convex functions but our experiments showed that a better solution for this problem is yet to be found.

Section 2 briefly mentions the techniques (hill climbing, *bumblebees* algorithm) we use in our tests and introduces the multiagent systems and the social phenomena in informatics in short. Section 3 provides the general use of the *bumblebees* algorithm and describes the optimization strategy in it. Section 4 presents the experimental results of the comparison and expresses the details of the inputs of the proposed algorithm. Section 5 discusses the obtained results and informs about the further work on the algorithm as the goal is to improve our initial experimental results.

2. Related work

A swarm of bees interacts with the whole beehive in a similar way to a multiagent system [10]. Every agent is an active agent as it seeks for food and every agent has information only about its solution for the optimization problem. Although, they share some information in the nest about the outside world, in our cases, especially about the food sources. We may say at first sight that the bee community acts randomly when seeking for food but the opposite of that is the truth. In [8] and [5], a bee that has found food, performs a "dance" for the other bees who are currently in the hive. The longer is the performance, the better quality has a food source which was found by the performing bee. Because of that process, bees in the audience can easily decide where to go to find food. That means the system (swarm of bees) has a useful knowledge about the quality of the food sources and that, in the end, helps them to survive.

However, the algorithm which we want to slightly modify and compare it with hill climbing, is the *bumblebees* algorithm [3]. It was derived from the *angels & mortals* [2] algorithm and, once again, the motivation of the agents in the artificial world is to acquire food. This algorithm is a simplified version of the food acquiring, whereas the bees do not spread the quality of the food source, the only thing they do is that they save the parameters of the source for the other bees in the nest. *Bumblebees* algorithm was tested on a classical NP-complete problem, the k-coloring of a graph both in [3] and [1]. In the latter, though, there was a successful experiment with a tabu search subroutine in the main algorithm.

Our decision was not to test *bumblebees* on the same optimization problem, but compare it with hill climbing on a non-convex mathematic function to find an optimal, global solution to the task. Hill climbing algorithm [7] can be designed in many ways. The simplest version is not a stochastic representation. It looks for the surrounding points and if there is a better solution, then it is the new midpoint in the next iteration. Its disadvantage is that there is no such technique that guarantees that we can get out of the environment of a local minima. If we choose small surroundings, we will never find a better solution in the next iteration because we stick in the area of a local minima. When a hard task occurs, such as a non-convex function with several local minimas and one global solution, there would be a need for a huge size to define the surroundings for one point. As it is not an ideal way to solve the problem, better results can be reached thanks to mutation. It provides stochastic outputs for the algorithm and we should not stick in the area of a local minima anymore. An even better way to reach the global solution in little time is to implement a parallel version of a stochastic hill climbing algorithm. When it is parallel, a few 'hill climbers' are being dropped into different positions as they look for a better solution in every iteration. Because of the fact that they are in different areas, there is a bigger chance that better results are reached sooner. Computationally it is not a harder task so the main comparison in our experiments is between the parallel stochastic hill climbing and the *bumblebees* algorithm.

In the next section, we describe the details of our proposed algorithm, which we want to compare with the hill climbing

3. The algorithm

Our algorithm somewhat differs from the algorithm introduced in [3]. We discuss the flow of our algorithm and concentrate mainly on the novelties in it. Our method begins in the same way as the cited algorithm. Firstly, the artificial world is generated, then the food sources are being generated all over the world along with the nest. As for the bumblebee population, their starting position is in the nest and exactly one of them moves out of the nest each generation. Before that, a random solution is assigned for every bumblebee and a lifespan based on that solution is being calculated for them, too. The basic structure of the algorithm can be seen in Algorithm 1.

In our case, *lifespans* are calculated in dependency of the population, but to speed-up the process of a new bumblebee's birth we made the decision to replace those ones who end their lives in the artificial world after their lifespan reaches zero. Lifespans depend not just on the quality of the solution, they rely on the size of the aforementioned population as well. The calculation of it is given by the next formula:

Algorithm 1. Our bumblebees algorithm

```
    function BUMBLEBEES($beeNum$ , $foodNum$ , $m$ , $n$, $t$, $t_{max}$)
        generateWorld($m \times n$)
        generateFood($foodNum$)
 4:     generateNest()
        generateBumblebees($beeNum$)
        for all Bumblebee do
            assignRandomSolution() & assignLifespan()
 8:         storeBestSolutions()
        end for
        while $t \leq t_{max}$ do        ▷ $t_{max}$ as the maximum number of generations
            $t = t + 1$
12:         decreaseLifespans()
            replaceBumblebees($best\_solutions$)
            moveBumblebees
            if $BumblebeePosition \leftarrow food$ then
16:             increaseLifeSpan()
                decreaseFoodCounter()
                moveBumblebeeToNest()
                sendFoodSourceToQueen()
20:         end if
            mutateBumblebeeSolutions()    ▷ mutate them with different techniques
        end while
        return bestSolution
24: end function
```

$$lifespan = 100\lfloor\frac{fitness}{population\_size}\rfloor, \qquad (1)$$

where the conditions of the fitness function can be described as follows:

$$fitness \leq population\_size. \qquad (2)$$

That implies that the lifespans of the bumblebees are assigned in order. In future tests, a lifespan can be, at most, ten times lesser than the number of the generations in the algorithm. The very bottom of the solutions, though, will have a minimal lifespan value so we can agily replace the worst solutions for the best at the start of the algorithm. Eventually, bumblebees representing the worst solutions with a zero lifespan value are being replaced shortly for the best solutions the algorithm had in its entire history. Namely, the size of the population never changes in the algorithm and we simply replace the old or weak ones. This design experiments with a technique that does not recalculate the lifespans every generation, only for the very first occasion. We can assume that if an adequate number of best solutions is stored throughout the algorithm, then missing out on recalculating the lifespans should not cause us any problem. Given that the mutation rate, which stands for the possibility of mutating one bit, is usually set to a low value in these types of algorithms, we do not feel the importance of recalculating the lifespans each iteration, as the solutions will not diverge drastically. The best way to have more new births is to reckon a lifespan that does not set its maximum value too big. Naturally, if there is a replacement of a bumblebee, one of the best solutions is assigned for the newly born representative with the biggest possible lifespan value. Lifespans are being decreased each generation by one unit.

Movement of the bumblebees is worked out in the same manner as it was in the original propose of the algorithm. One bumblebee moves out of the nest each generation and randomly occupies one of the 24 nearest cells. If it finds food, it increases its lifespan by two units, whilst the food counter is decreased by one unit. The primary value of the food counter is 5 and when it reaches zero, then a new food source is generated randomly in the world. An agent, which has found food, stores the parameters of the source for the rest of the population that stay in the nest. It is always known how many bumblebees approached the source that has been already discovered so there is no misinterpretation between them while searching for food. When the bumblebee knows where it will go, its movement remains still a bit random. The direction of its movement will be always deterministic, but the size of the step is calculated still randomly.

Final thing is the *mutation* of the solutions that are represented by the bumblebee population. As we already need a sorting function to assign the lifespans for the bumblebees at the start of the algorithm, we can manipulate with the mutation rate after another sorting in every iteration. For example, we can mutate worse solutions with a bigger possibility than the better ones. We can choose whether we want to mutate the higher or the lower bits. Taking into account our comparison with the hill climbing algorithm, in *bumblebees* we have more of a freedom to mutate each bits to find better solution than we have in the other instance, because hill climbing does not give us any chance to mutate solutions in any other way but with one possibility rate.

To get a more complex view for our research, our decision is to use the proposed algorithm in two versions. Firstly, we go through the state space in a manner we call 'random walk', that means that any mutated solution will be the saved solution for the agent, even if it was worse than the original one. The other version is, when the algorithm works in a way where every solution is updated only if the new, mutated solution is a more qualitative solution. In other words, we always have a knowledge of the best solution, which was found by a bumblebee. The reason we test the algorithm in both versions is because our testing functions differ and we want to know, whether the 'random walk' experiment does help when we have a typical task where we might stuck in a local minima.

In short, we can put down that our algorithm differs from [3] in terms of the mutation process and, in particular, the lifespan calculating which we discuss fully as aforesaid. The main difference is that we do not recalculate lifespans each generation, only at the very start. Also, we have set a permanent population size, contrary to [3]. The initiative parts of the algorithm remain the same just as the movement of the bumblebees.

4. Experimental results

We compare our algorithm with two versions of hill climbing algorithm on a proper non-convex function, the Rastrigin function. In our study, let n be number of the dimensions, then the first function is defined by [4]:

$$f_1(x_1, x_2, ...x_n) = A \cdot n + \sum_{i=1}^{n} x_i^2 - A\,cos(2\pi x_i), \quad (3)$$

where $A = 10$ and $x \in [-5.12;5.12]$. It has a global minimum at $x_i = 0$ where $f_1(x_1,x_2,...x_n) = 0$. Also, it has a large number of local minimas but the aim of our optimization is to find the global solution. The second function that we used for the purpose of testing the algorithm is defined by [6]:

$$f_2(x) = 0.993851231 + e^{-0.01x^2} sin(10x)cos(8x), \quad (4)$$

where $x \in [-10,10]$. Besides prevalent local minimas, it has a global minimum at $x = -0.7853$ where $f_2(x) = 5.69\times 10^{-11}$. To generalise this function, we define it by [6]:

$$F_2(x_1, x_2, ...x_n) = \sum_{i=1}^{n} f_1(x_i), \quad (5)$$

where n represents the number of the dimensions. The global minimum is at $x_i = -0.7853$. Our third function is the Schwefel function, and it is defined by [4]:

$$f_3(x_1, x_2, ...x_n) = A \cdot n + \sum_{i=1}^{n} x_i^2 - A\,cos(2\pi x_i), \quad (6)$$

where $x_i \in [-500;500]$ and n is the number of the dimensions. It has a global minimum at $x_i = 420.9687$, where $f_3(x_1,x_2,...x_n) = 0$.

We describe the input parameters of the *bumblebee* algorithm as follows. A 20×20 world is generated with *40* food sources. The population, alike the world and the food sources, remains of the same size throughout the whole process. We have set the population to *50*, *100* and *200* sequentially in our tests for all three functions. To get more accurate results, we used *31* bits to represent the solutions in $f_1(x)$. On the other hand, in $F_2(x)$ and $f_3(x)$ it is enough to generate a binary vector that consists of *16* bits. Thus, random solutions are being represented by 2^k numbers, where k is the number of bits and the next needed calculation is to interpret it as a real number. Afterwards, we get x and can easily compute the function value for the real number from the already given interval. The maximum value of a bumblebee's lifespan matches with the parameter described in the previous section and its value is set to 100. A new-born bumblebee receives that number. The number of generations (t_{max}) is *1000*. The algorithm saves the *10* best solutions, these values are handed to the new members of the population. The information about the best solutions is updated once a generation. Unlike the original algorithm, we did not recalculate the lifespans of the bumblebees after mutation.

We used identical number of inputs in the hill climbing algorithms. Therefore, in a simple stochastic hill climbing, the number of the generated surroundings was *50* and *100* and t_{max} is 1000, naturally. In parallel hill climbing, where the hill climbers work independently at random places, we tested with the same population size as in *bumblebees*, who expect better solutions thanks to one newly generated surrounding each round. The number of the iterations remains *1000*.

Table 1: Summary of our comparison between the parallel hill climbing and the bumblebees algorithm

Functions, dimensions and success rate in %												
	$f_1(x)$				$F_2(x)$				$f_3(x)$			
	$1D$	$2D$	$3D$	$4D$	$1D$	$2D$	$3D$	$4D$	$1D$	$2D$	$3D$	$4D$
pHC	100	100	**97.4**	2.6	**99.8**	**68**	4.9	0.1	**100**	**76.6**	4.8	0
BB	100	100	89.6	**5**	93.2	24.3	**7.4**	**1.25**	94.8	52.8	**18.2**	**2.38**

Possibility of mutating was set to a rate that expects one bit is being mutated and the others remain the same. In *bumblebees*, however, the worse solutions were mutated by bigger possibility. Higher bits have this rate twice as big as the common routine described above.

Table 1 describes the summarized results we got after testing our algorithm in comparison with the parallel hill climbing. These results show only that version of the *bumblebees* algorithm, where we did not experiment with the 'random walk' technique. It clearly shows that when we incremented the number of the dimensions (that means exponentionally a bigger state space), we got better results, so our technique with the lifespan calculation and the food acquisition worked out in a solid manner. Furthermore, our initial experiments showed that the process of the food acquisition optimizes the results, as we did study our algorithm in some tests with a modified *bumblebees* algorithm, when there was no such similar process as the movement and the food acquisition. One-dimensional tests took part with a population size set to *50*, *100* agents appeared on 2D and 3D tests, while on four-dimensional tests we used population *200*.

Table 2 shows the results we obtained after implementing all three algorithms. In this test, we used the 'random walk' experiment for our proposed method. When the population size is *50* or *100*, the parametres for the two hill climbing algorithms were adequately chosen. As stated above, t_{max}, that is the number of generations was constantly *1000*. Simple and parallel stochastic hill climbing performed faster then the *bumblebees* algorithm, but the finding of the global minima was variant when we tested it on the two functions. Whilst the parallel hill climbing algorithm appeared to be the same on the $f_1(x)$ (Rastrigin) function, our *bumblebees* algorithm with the 'random walk' technique outperformed the simple hill climbing algorithm on $F_2(x)$, on the other hand, it did not precede the parallel hill climbing. Though, it proved to be the better solution on both occasions in comparison with the simple stochastic hill climbing. Although, the variation with the possibilites of the mutation did not help us to get better results, nor it worsen the outcome.

All the implementation was programmed in C language and executed on an Intel Core Duo CPU at 2.80Ghz.

5. Conclusions

Our target in this work was a contribution to optimization solving algorithms procured by a relatively young method inspired by the social insect behaviour. Results show that a respective improvement was attained and that there is room for further consideration.

We took notice of the fact that while hill climbing algorithms, especially the simple version of it, were caught sticking in a local minima far from the global solution on a few occasions, until then if our *bumblebees* algorithm did not find the global minimum of a function, it was still in its very proximity. That means that a proper way to determine the global solution when we are near to it would be sufficient. Given the state that we always have a knowledge about the current best solutions, manipulate with them could be an ideal answer to our problem. We can almost guarantee that in the last iteration the very best solutions are a nearby value to the global minimum. Consequently, if we calculate the surroundings of the latest best solutions, potentially we get the global minima. Even a tabu search principle may work if we want to prevent the imperfection which is causing that we always end up finding the same function values.

Table 2: Results for the comparison between a simple (sHC) and parallel hill climbing (pHC) with the *bumblebees* (B) algorithm on two selected non-convex functions

$population = 50$						
	$time$			success rate in %		
	sHC	pHC	B	sHC	pHC	B
$f_1(x)$	$0.078s$	$0.062s$	0.086s	89	100	100
$F_2(x)$	$0.062s$	$0.030s$	0.094s	42	99.8	87.2

$population = 100$						
	$time$			success rate in %		
	sHC	pHC	B	sHC	pHC	B
$f_1(x)$	$0.109s$	$0.125s$	0.140s	94.2	100	100
$F_2(x)$	$0.094s$	$0.110s$	0.156s	48.5	100	100

Additionally, we are in need of a further study to find out whether that is it necessary to build into the algorithm the movement of the population or can we cope without it in these kind of performance test problems. As other works showed in the recent past, that technique (the food acquisition) along with the natality and mortality can improve the results on typical NP-complete problems.

To sum it up, our further research should include the ideas we proposed above to improve the performance of the algorithm and to demonstrate whether all of the steps in our *bumblebees* are requisite.

References

[1] D. Chalupa. An optimization strategy based on social insect behavior. In M. Bieliková, I. Černá, T. Gyimóthy, J. Hromkovič, K. Jeffery, R. Královič, M. Vukolić, and S. Wolf, editors, *SOFSEM 2011. Student Research Forum*, pages 35–50, 2011.

[2] F. Comellas and R. Gallegos. Angels and mortals: A new combinatorial optimization algorithm. In W. Hart, J. Smith, and N. Krasnogor, editors, *Recent Advances in Memetic Algorithms*, volume 166 of *Studies in Fuzziness and Soft Computing*, pages 397–405. Springer Berlin Heidelberg, 2005.

[3] F. Comellas and J. Martinez-Navarro. Bumblebees: a multiagent combinatorial optimization algorithm inspired by social insect behaviour. In *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, GEC '09, pages 811–814, New York, NY, USA, 2009. ACM.

[4] J. M. Dieterich and B. Hartke. Empirical review of standard benchmark functions using evolutionary global optimization. *CoRR*, abs/1207.4318, 2012.

[5] A. B. Ezzeddine. Web information retrrieval inspired by social insect behaviour. In M. Bieliková and P. Návrat, editors, *Information Sciences and Technologies. Bulletin of the ACM Slovakia*, pages 93–100, 2011.

[6] V. Kvasnička, J. Pospíchal, and P. Tiňo. *Evolučné algoritmy*. STU, Bratislava, Slovakia, 2000.

[7] A. W. Moore. Iterative improvement search, 2002, available from the web in February 2013.

[8] P. Navrat, A. B. Ezzeddine, L. Jastrzemska, and T. Jelinek. Exploring the bee hive metaphor as a model for problem solving: Search, optimisation, and more. In Z.-U.-H. Usmani, editor, *Web Intelligence and Intelligent Agents*. InTech, 2010.

[9] A. Neme and S. Hernández. Algorithms inspired in social phenomena. In R. Chiong, editor, *Nature-Inspired Algorithms for Optimisation*, volume 193 of *Studies in Computational Intelligence*, pages 369–387. Springer Berlin / Heidelberg, 2009.

[10] K. P. Sycara. Multiagent systems, 1998, available from web in February 2013.